Adaptive Voltage Scaling in a Heterogeneous FPGA Device with Memory and Logic In-situ Detectors

Jose Nunez-Yanez, Department of Electrical and Electronic Engineering,

University of Bristol, MVB Building, Woodland Road,

BS8 1UB, Bristol, UK

j.l.nunez-yanez@bristol.ac.uk

Abstract. This paper presents an enhanced tool flow and hardware to allow a host CPU to exploit the timing margins available on a FPGA fabric to improve its performance or reduce its energy and power requirements. Two different case studies are considered to demonstrate the performance gains and energy reduction possible in realistic scenarios. The first case study presents a video fusion system with hardware acceleration. The video fusion application is based on Dual-Tree Complex Wavelet Transforms (DT-CWT) that are mapped to a hardware accelerator using high-level synthesis tools. The hardware netlist is processed and in-situ detectors are automatically added to monitor and pre-detect timing failures occurring in the critical path flip-flops. In the second case study the tool flow is extended to support cases where the critical paths terminate in memory blocks with internal registers hidden from the user. A soft-core multiprocessor implemented in the FPGA is used to illustrate the additional challenges and proposed solution. In both cases the host CPU can control the voltage and frequency of the FPGA and compute to the performance or energy limit obtaining around 70% increase in performance or reduction in energy. Intermediate solutions that trade different levels of performance for energy are also possible. The system exhibits excellent energy proportional computing characteristics and can adapt its operating point to complete a task within a given time budget so that only the minimum level of energy is used.

1 Introduction

Heterogeneous computing is seen as a path forward to deliver the energy and performance improvements needed over the next decade. In heterogeneous architectures specialized hardware units accelerate a complex task. A good example of this trend is the introduction of GPUs (Graphics Processing Units) for general purpose computing combined with multicore CPUs. FPGAs (Field Programmable Gate Arrays) are an alternative high performance technology that offers bit-level parallel computing in contrast with the word-level parallelism deployed in GPUs and CPUs. Recently, the traditional low-level programming languages used in FPGA design have started being replaced with high-level languages such as C++ successfully. This new programming models and the acceleration capabilities of FPGAs for certain tasks have increased the interest in computing systems that combine CPUs and FPGAs with significant efforts done, for example, by Intel with their HARP program [1], Microsoft with their Catapult framework [2] and IBM introducing coherent ports for FPGA acceleration in OpenPower [3]. In this research we consider a heterogeneous device that includes the CPU and FPGA devices in the same die and explore the additional performance and energy possible when a closed-loop system is able to monitor and adjust the voltage and frequency parameters of the FPGA. The proposed system is based on the Zynq System-on-Chip and it runs under the Linux OS with a customized kernel level Linux driver. The main contributions of this paper are:

1. We extend the Elongate [4] energy proportional computing framework with in-situ detectors that work with IP cores with different types of critical paths. The main novelty is that while the original work presented in [4] and [19] was limited to critical paths with flip-flops as the end-points a new type of detector and detector insertion flow is presented able to handle cases in which the end-points are located in BlockRAMs that have no user-accessible flip-flops.
2. The application of the framework to a video fusion system created with high-level synthesis tools originally performed in [20] is extended with a second case study. This second case study corresponds to a multiprocessor system with Microblaze soft-cores forming an overlay architecture. In this system we

investigate the energy requirements of run fast and clock gate with run slow configurations that are voltage and frequency scale in the FPGA.

The rest of this paper is organized as follows. Section II discusses related work in this research area. Section III presents the energy proportional implementation flow and compares the detectors created for different critical path endpoints. Section IV introduces the overall hardware architecture with a customized kernel level Linux driver to implement data movers and interrupt generation. Section V presents details on the user logic for both test cases. Section VI investigates the performance and power of both test cases applying the voltage and frequency scaling features. Section VII extends this analysis to include energy calculations. Finally, section VIII concludes the paper.

2 Related Work

In order to identify ways of reducing the power consumption in FPGAs, some research has focused on developing new FPGA architectures implementing multi-threshold voltage, multi-Vdd and power gating techniques [6-9]. These techniques are designed to be used in new architectures and devices. Other strategies have proposed modifying the map and place&route algorithms to provide power aware implementations [10-12]. Similarly, main FPGA manufacturers currently offer switches in their tools that optimize the synthesis and implementation runs for power in addition to performance or area. Recently FPGA manufacturers have also started investigating the topic of voltage and frequency adaptation. Xilinx supports the possibility of using lower voltage levels to save power in their latest family implementing a type of static voltage scaling in [13]. The voltage identification *VID* bit available in Virtex-7 allows some devices to operate at 0.9 V instead of the nominal 1 V maintaining nominal performance. During testing, devices that can maintain nominal performance at 0.9 V are programmed with the voltage identification bit set to 1. A board capable of using this feature can read the voltage identification bit and if active can lower the supply to 0.9 V reducing power by around 30%. Altera offers a similar technology with the *SmartVoltage ID* bit [14] and future devices will take this concept further with *Vcc PowerManager*. These chips can operate at either the standard Vcc voltage or a set lower voltage level by lowering the frequency. This feature can reduce total power by up to 40 percent and is suitable when maximum performance is not required all the time. These techniques are open-loop in the sense that valid working points are defined at fabrication time and not detected at run-time as in this research.

Run-time dynamic voltage scaling strategy for commercial FPGAs that aims to minimise power consumption for a giving task is presented in [15]. In this methodology, the voltage of the FPGA is controlled by a power supply that can vary the internal voltage of the FPGA. For a given task, the lowest supply voltage of operation is experimentally derived and at run-time, voltage is adjusted to operate at this critical point. A logic delay measurement circuit is used with an external computer as a feedback control input to adjust the internal voltage of the FPGA (VCCINT) at intervals of 200ms. With this approach, the authors demonstrate power savings from 4% to 54% from the VCCINT supply. The experiments are performed on the Xilinx Virtex 300E-8 device fabricated on a 180nm process technology. The logic delay measurement circuit (LDMC) is an essential part of the system because it is used to measure the device and environmental variation of the critical path of the functionality implemented in the FPGA and it is therefore used to characterise the effects of voltage scaling and provide feedback to the control system. This work is mainly presented as a proof of concept of the power saving capabilities of dynamic voltage scaling on readily available commercial FPGAs and therefore does not focus on efficient implementation strategies to deliver energy and overheads minimisation.

A similar approach is also demonstrated in [16]. In this case a dynamic voltage scaling strategy is proposed to minimise energy consumption of an FPGA based processing element, by adjusting first the voltage, then searching for a suitable frequency at which to operate. Again, in this approach, first the critical path of the task under test is identified, then a logic delay measurement circuit is used to track the critical point of operation as voltage and frequency are scaled. Significant savings in power and energy are measured as voltage is scaled from its nominal value of 1.2V down to its limit of 0.9V. Beyond this point, the system fails. The experiments were carried out on a Xilinx ML402 evaluation board. The XC4VSX35-FF668-10C FPGA used in this board is fabricated in a 90 nm process and energy savings of up to 60% are presented. The previously discussed efforts are based on the deployment of delay lines calibrated according to the critical path of the main circuit. This calibra-

tion is cumbersome and it could lead to miss tracking due to, for example, the different locations of the delay line and the critical paths of the circuit having different temperature profiles. In-situ detectors located at the end of the critical paths remove the need for calibration. This technology has been demonstrated in custom processor designs such as those based around ARM Razor [17]. Razor allows timing errors to occur in the main circuit which are detected and corrected re-executing failed instructions. The latest incarnation of Razor uses a highly optimized flip-flop structure able to detect late transitions that could lead to errors in the flip-flops located in the critical paths. The voltage supply is lower from a nominal voltage of 1.2V (0.13µm CMOS) for a processor design based on the Alpha microarchitecture observing approximately 33% reduction in energy dissipation with a constant error rate of 0.04%. The Razor technology requires changes in the microarchitecture of the processor and it cannot be easily applied to other non-processor based designs. Application specific circuits presented in [21] for image processing have also shown the potential of AVS at reducing both dynamic and leakage power. In [21] a novel simulation methodology based on Markov models is developed to account for the variability of the delay paths depending on input patterns. The approach is validated using a DCT (Discrete Cosine Transform) engine implemented in 65 nm low-power CMOS technology. Our previous work has demonstrated the power and energy benefits of deploying voltage scaling using in-situ detectors in commercial FPGAs in [19] and [4]. In this paper we show that the flow is compatible with the latest generation of FPGA tools that deploy high-level synthesis flows (i.e. Vivado HLS) and IP integration (i.e. Vivado IPI). We also extend the flow to support new circuits that show different types of critical path end-points not limited to flip-flops. The paper in [20] presented initial results around the video fusion application and Vivado HLS and this is extended in this work with a new tool flow and detector type that enables having internal memories as end-points of the critical paths.

Significant to this research is the FPGA-focused voltage and frequency scaling work done in [18] which uses an online slack measurement (OSM) technique. The OSM method uses direct timing measurement of the application circuit to respond to variation, temperature, and degradation. It also deploys shadow registers that are clocked with a different clock phase. The phase of this clock constantly adjust to determine the point in which discrepancies between the main flip-flop and the shadow flip-flop take place. The shadow registers are not located in the same logic cells of the main flip-flop so a recalibration technique is performed offline to remove the variable delays introduced by the variable placement and routing. Similarly to our previous work it can only be applied to logic circuits since it relies on comparison between the values of the main flip-flop and the shadow flip-flop. In contrast, our approach does not require recalibration and does not perform online measurements since it relies on placing the shadow register in the same slice as the same flip-flop. Our approach is guided by the static timing analysis tool although it does not use absolute timing values but it needs to know how paths are organised in terms of timing. Both approaches target different vendors (i.e. Altera in [18] and Xilinx in this work) and rely in microarchitecture features available in the respective devices such as slices that can accommodate both flip-flops or highly accurate phase control that can lock fast enough to avoid being a significant overhead. For this reason it will be quite challenging to port each methodology to a different vendor. Each approach has logic overheads that are highly dependent on the benchmark circuit and the number of shadow registers inserted.

3 Energy Proportional Implementation Flow

3.1 Flow overview

The energy proportional implementation flow introduces the in-situ detectors in the design netlist guided by post place&route timing information. The core of the flow is the Elongate tool [4] that transforms the original design netlist into a new netlist with identical functionality and additional power management IP and in-situ detectors. Fig. 1 shows the overall flow that can be decomposed into three distinct phases indicated with numbers 1, 2 and 3 in Fig. 1. During the first phase the original netlist goes through a full implementation run to obtain post place&route timing data in the form of a text file. In the second stage the Elongate tool takes as input the obtained timing data, the original netlist and Elongate component library that describes the power management core and in-situ detectors and produces the new power adaptive netlist. The third stage consists of a final implementation run of the power adaptive netlist to obtain the device bitstream ready to be downloaded in the device. This third stage is done using the incremental place&route available in the Vivado flow to minimize

changes to the implementation caused by the addition of the in-situ detectors and additional logic. The incremental flow enables the reutilization of approximately 98% of the available place&route information.

The Elongate framework can work with sources in VHDL/Verilog or with high-level descriptions since its input is a technology mapped netlist obtained from these descriptions. In the video fusion case study the input into the flow in Fig. 1 is a C++ description of the forward and inverse wavelet filters used in the DT-CWT that are initially compiled with Vivado HLS tools. This netlist is then synthesized to obtain a new VHDL netlist based on the implementation primitives available in the target technology (e.g. LUTs, flip-flops, etc.). These initial synthesis steps are required to obtain the netlist that will be processed by Elongate. The need for this initial preprocessing is because the Elongate transformation does not take place at source level directly. The reason is that slight changes in the source can have a large effect on timing and also because it is possible to annotate the critical paths found after static timing analysis with the physical flip-flops in the netlist. The timing information enables Elongate to replace the end-point flip-flops in the critical paths with new soft-macro flip-flops that incorporate the in-situ detection logic. These soft-macro flip-flops are VHDL files that implement the logic shown in Fig. 2 with the addition of placement constraints so the netlist primitives are placed and routed exactly as shown in the figure. The internal configuration of this in-situ detection logic for logic detectors is shown in Fig. 2.a and for memory detectors is shown in Fig. 2.b. Each primitive flip-flop component in the technology library has a corresponding soft-macro flip-flop stored in the Elongate component library with identical functionality. Part of the user constraints input to Elongate indicate the level of coverage requested for the critical paths in the design. The coverage must be sufficient so that the critical paths of the final design have as endpoints the newly inserted soft macro flip-flops.

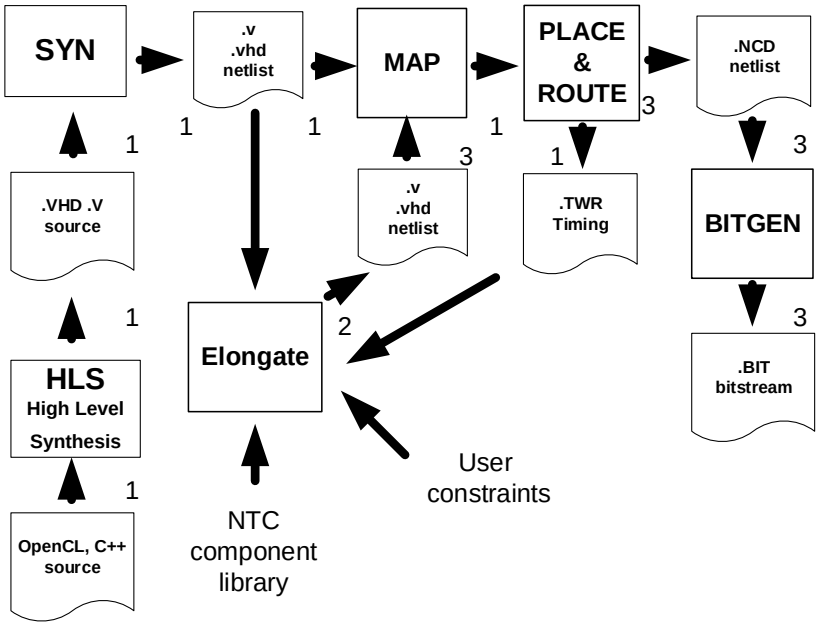

Fig. 1. Energy Proportional Implementation Flow

In the case of the Microblaze system the initial timing analysis reveals that the critical paths extend from the Microblaze logic to embedded BlockRAMs. To have the embedded BlockRAMs as endpoints requires a different type of detector that is presented in the next section.

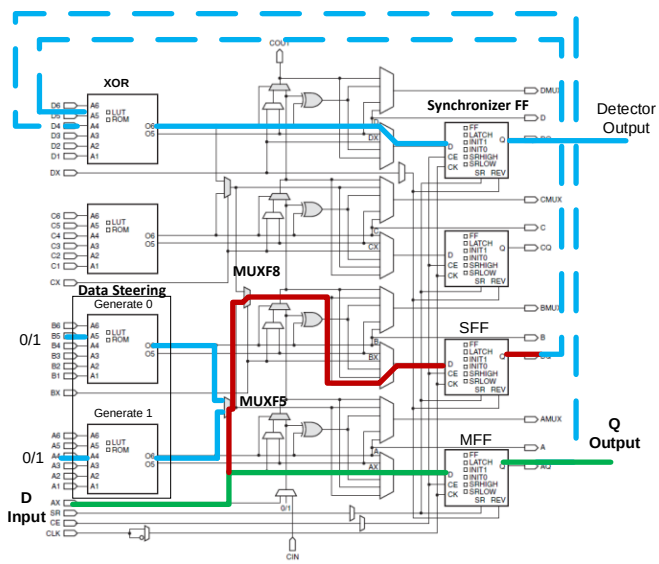

Fig. 2.a. Logic timing detector.

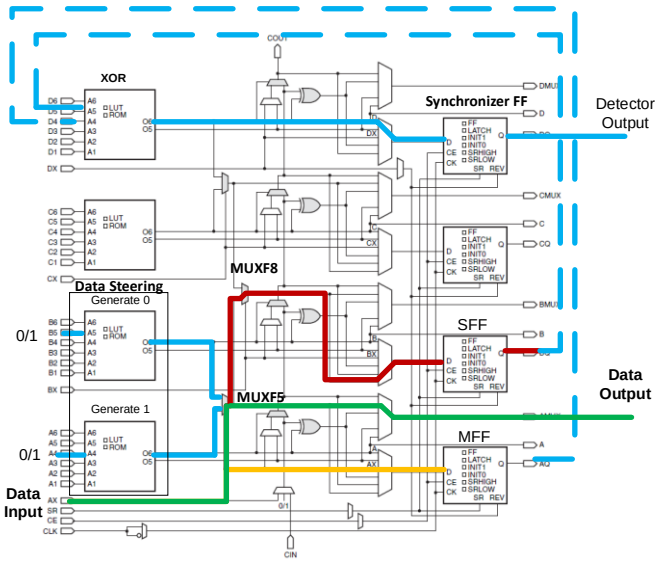

Fig. 2.b. Memory timing detector.

3.2 Analysis of the timing detectors

Fig. 2 shows a comparison between the memory and logic detectors. In the logic detector the data enters through the D input and it follows the lower path to the MFF (main flip-flop) that maintains the same functionality as the original flip-flop it replaces. The data steering logic generates two bits set to zero and one that are input to the multiplexor MUXF5. The effect is that MUXF5 passes the value of the signal input into its control port which is the D input to the output of the multiplexor. The output of MUXF5 is then routed to the lower input of multiplexor MUXF8 that is configured to always select this input as its output. This creates a path through MUXF8 that ends in SFF (slow flip-flop) and that is slightly slower than the original path. The path to SFF from the D input is slightly slower than the path from the D input into MFF because there is more logic involved and this can be verified using the static timing analysis tools provided by the vendor.

As the frequency that clocks the flip-flops becomes critical for a given voltage level discrepancies between SFF and MFF are detected by the XOR and forwarded to the controlling logic (not shown). The changes in frequency happen to a level of granularity that SFF fails timing before MFF. The synchronizer FF at the top of Fig. 2.a removes possible metastability originating in SFF. This approach has already been presented in [19] but a limitation can be found when the critical path end-points are located in the internal memory blocks. These memories called BlockRAMs are all fully synchronous and both the address and data inputs are registered internally. The data outputs are available in the next clock cycle. If we were to use the same type of logic sensor in front of the memory inputs we would introduce an additional cycle delay that would alter the timing of the circuit and invalidate its functionality. To solve this problem the detector circuit shown in Fig. 2.b is proposed. In this figure the data input flows directly through MUXF5 to the data output and this data output is the one connected to the BlockRAM. This configuration does not affect the timing of the BlockRAM and inside the detector the data input is connected to both SFF and MFF to detect discrepancies. Without further modifications timing errors will

take place in the BlockRAM before they are detected since the path to the BlockRAM is longer than the path to SFF and MFF. To address this problem we add an additional clock *Clk2* connected as shown in Fig. 3. In this figure *Clk1* represents the original clock signal while *Clk2* is the clock signal used by the detector that is locked to *Clk1* with the same clock frequency but a different clock phase.

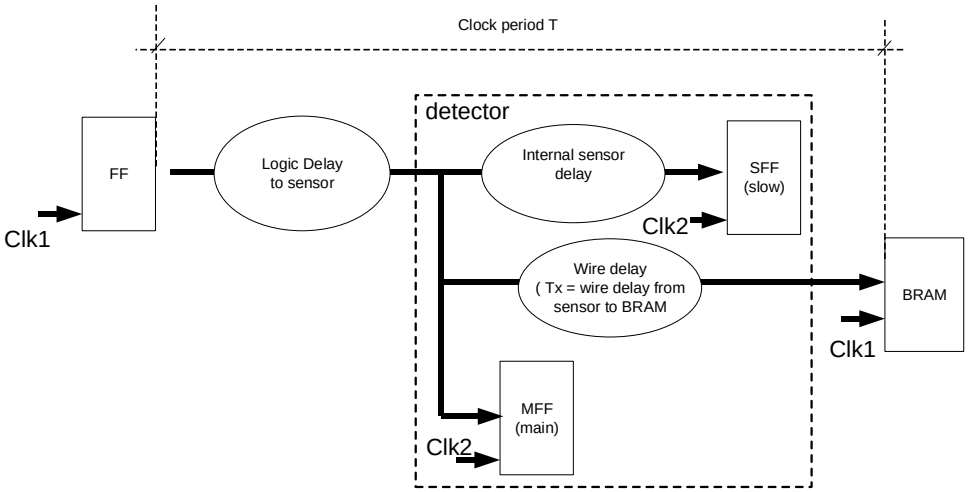

Fig.3. Memory sensor schematic located in critical path

Fig. 4 shows the relationship between *Clk1* and *Clk2* and how the required phase for *Clk2* can be obtained. *T* is the period of *Clk1* which is the same as *Clk2*. The *Tx* delay represents the additional delay from the sensor to the BRAM inputs. Ideally the *Tx* delay should be zero so that the sensor is perfectly located in front of the BRAM and can detect timing errors before they affect the BRAM. This is not possible since this level of timing control is not feasible in a FPGA device and also not practical because generally there are multiple BRAMs driven by the same clock signal and the sensor will need to be replicated for each BRAM. Instead the *Tx* delay is compensated by adjusting the phase of *Clk2*. This adjustment reduces the time available for the signals to propagate from the rising edge of *Clk1* to the sensor flip-flops driven by the rising edge of *Clk2*. The end result is that the phase adjustment creates critical paths to the flip-flops that fail earlier than the functional paths to the BRAM. These timing errors are detected with differences between SFF and MFF and communicated to a control logic that takes the necessary actions. Fig. 4 shows how the phase adjustment in *Clk2* compensates for T_x. The new phase of *Clk2* can be obtained with this simple relation $Clk2\Phi = 360 * (1 - T_x/T)$. For example, If $T = 2*T_x$ the new phase $Clk2\Phi$ is 180 and the time available for signals to propagate to MFF is effectively half a clock period *T*.

Notice that if, for example, the path slows down by 20% due to temperature, process variations or aging then the new $Clk2\Phi = 360 * (1 - 1.2*T_x/1.2*T)$ is still the same and constant since both time variables are affected by the same delay. It is reasonable to assume that the additional delay will affect both values since the paths are physically collocated and multiple paths are protected. This is more robust than using delay lines since their location can be quite different from the critical path location and typically only one or a small number of them are used across the device. Since it is not practical to locate delays lines near all the possible critical paths there is a higher chance of calibration errors that do not affect the proposed approach. Additionally, in an implementation based on delay lines the proportion of wire segments and LUT elements in the delay line will be different from those on the real critical path and could be affected by variability in a different way while in the proposed approach the detection circuit and the real critical path shared the same wire segments and logic elements by design and this increases the robustness of the approach to variability.

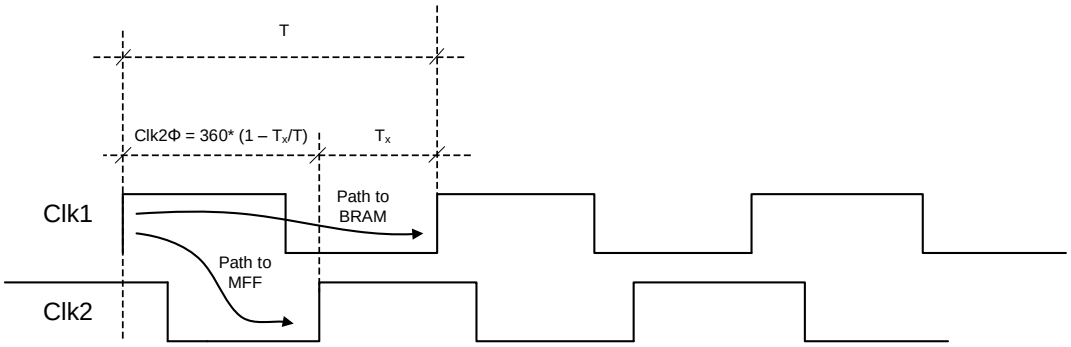

Fig. 4. Timing relationships between *Clk1* and *Clk2*

There are many possible *T* and *Tx* depending on how many critical paths are analysed. To be able to calculate the *Clk2Φ* a new routine has been added to the Elongate framework that processes the timing analysis files and extracts the larger *Tx* delay between memory and detectors. *Tx* can be obtained from the timing analysis files as the time difference between the delay of the path that finishes in the BlockRAM (clocked by *Clk1*) and the path that finishes in the MFF flip-flop (this path initial flip-flop is clocked by *Clk1* while MFF is clocked by *Clk2*). It then uses the larger *Tx* to obtain the necessary *Clk2Φ*. Notice that a large *Tx* will result in a smaller *Clk2Φ* but this does not necessarily mean that the performance of the circuit is reduced since this is determined by *T* which is independent of *Tx*. In summary, these timing relations imply that the memory sensor will fail timing before the BRAM because SFF is slower than MFF which itself thanks to the new *Clk2Φ* fails timing simultaneously to the BRAM. The performance of the circuit is independent of *Clk2Φ* because it is determined by *T* that is a characteristic of the circuit as the addition of clock skew, clock uncertainty, combinatorial delays and setup delays.

The flow to handle paths to BRAMs located outside the IP core is shown in Fig. 5 and it extends the original Elongate flow that adds in-situ detectors to logic critical paths as presented in [19]. Initially a full implementation is completed with both clocks in phase. Then the clocks are set 180 degrees out of phase that makes the paths from *Clk1* to *Clk2* critical and observable and a new timing report file is generated. The verify tool processes this new report and obtains *Tx* and *T* from the static timing analysis reports that are then used to calculate *Clk2Φ*. The clock phase is then modified and a new bitstream generated. It is important that the modification to the clock phase is done without creating new constraints and re-implementing the design. Otherwise the optimization process will alter the timing of the circuit resulting in an invalid analysis. An alternative way to achieve this objective could consist in declaring paths to the detectors as invalid so that the optimizer ignores them.

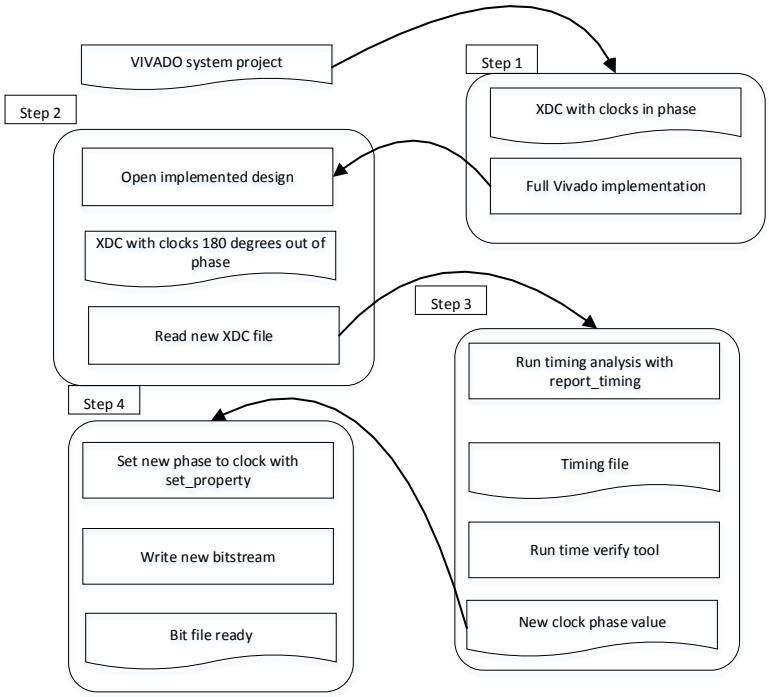

Fig. 5. Flow for in-situ memory sensor insertion

It is clear that the timing violations in the detectors are data dependent. In the current setup, the hardware runs for a time step long enough so that the probability that the critical paths are exercised increases. In the fusion application the time step is a whole frame. Otherwise there is the possibility that unsafe voltage or frequency scaling decisions are taken because the critical paths have not been exercised. A possibility of addressing this problem is to add logic that verifies that the critical paths are exercised before taking a scaling decision. Notice that it is not enough to check that the output of the end-point flip-flop is changing since these changes could be due to other paths that are not critical and connect to the same end-point flip-flop. The overhead of monitoring the whole path for changes will be considerably but this safer approach will be necessary if this technology was to be deployed in safety critical applications.

4 The Elongate System Architecture

The overall architecture is shown in Fig. 6 which contains two voltage domains corresponding to the programmable logic (PL) and processing system (PS). The PL voltage domain is the region dedicated to the user IP and the DFS (dynamic frequency scaler). The PS voltage domain contains the Cortex A9 processors, I2C controller, memory controller and additional peripherals. We have used the Zynq-based ZC702 Evaluation Board running Ubuntu Linux OS. The ZC702 board uses programmable power regulators and a PMBus compliant system controller to supply power to all the components present in the Zynq chip through a number of independent rails. Using the PMBus protocol it is possible to issue commands to the voltage regulators to write and read voltage, current and power information. The ARM processors accesses the PMBus through a voltage shifter and an I2C 1-to-8 switch present on the board. The initialization code must set the 1-to-8 switch to the PMBus channel before communication with the voltage regulators is possible. Once this initialize phase is completed the ARM processor performs voltage scaling and obtains power measurements using the PMbus protocol. This means that the DVS (Dynamic Voltage Scaling) unit originally presented in [19] has been removed since voltage scaling is done by the ARM directly using the I2C controller shown to the right of the figure. The DFS (Dynamic Frequency Scaler) shows the addition of a PLL whose clock input is connected to the clock output of the MMCM (Mixed Mode Clock Manager). The PLL is configured to generate the same clock frequency as the MMCM but to shift the phase as indicated by the value $Clk2\Phi$ obtained in section 3. The adaptation control state machine controls the MMCM to generate multiple clock frequencies and resets the PLL to allow it to relock at the differ-

ent clock frequencies. The control state machine generates up to 535 different frequencies between the ranges of 20 MHz and 400 MHz with fine increments. It constantly monitors the status of the user IP block and decides to increment or decrement frequencies depending on this status. The adaptation control logic also monitors device parameters such as temperature to verify that it remains below the allowed maximum. The user IP block is the user logic in charge of performing the useful computation that has been embedded with the logic and memory detectors. The detectors which are shown as EFF in the figure communicate through the EFF control with the adaptation control logic.

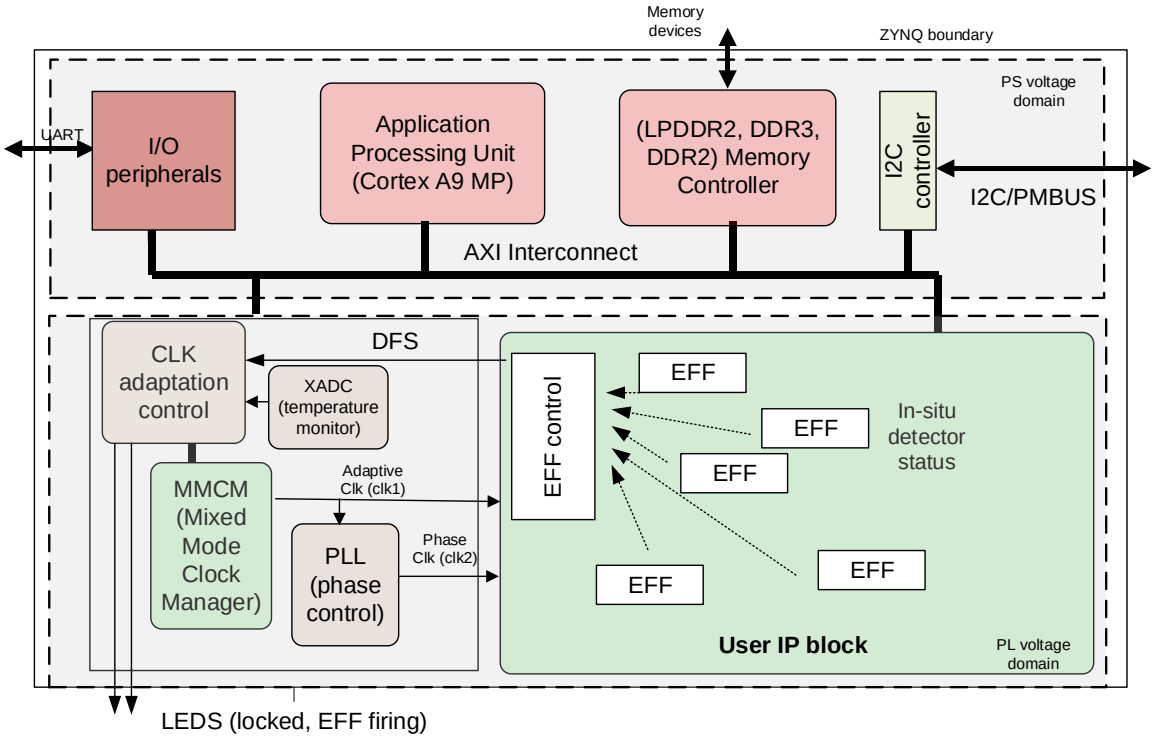

Fig. 6 Overview of the system Design

5 Video Fusion and Microblaze Multiprocessor Case Studies

5.1 Video fusion application

Video fusion can be considered a special case of image fusion when two or more frames of different video sources are fused together continuously into a single fused video. Image fusion can be performed based on wavelet transform techniques that achieve better signal to noise ratios and improved perception with no blocking artefacts. Among the available wavelet transforms the use of the Dual-Tree Complex Wavelet Transform (DT-CWT) has been shown to produce significant fusion quality improvement [22]. The algorithm described in [22] is used in this case study which consists in applying DT-CWT to infrared and visible frames, combining the obtaining coefficients using a fusion rule and then proceeding to perform the inverse DT-CWT for reconstruction. As a first step the whole fusion algorithm with the forward and inverse DT-CWTs is written in C++ and executed by the ARM Cortex A9 Processor available in the Zynq device. The profiling results indicate that the forward and inverse DT-CWT are the most compute and energy intensive tasks. Therefore, these parts of the algorithm are the ones selected for acceleration in the FPGA. The DT-CWT hardware needs an efficient method to move pixel data from the external memory. The general purpose GP AXI 32-bit ports available in Zynq are low-bandwidth ports due to the central interconnect switch and do not support bursts or DMA transfers. They do not obtain the require performance and every transfer requires around 25 clock cycles with the CPU moving the data itself. For this reason we created a custom DMA engine using the synthesis support of memcpy by Vivado HLS connected to the high-performance ACP (Accelerator Coherence Port) port available in the Zynq device. The code for Vivado HLS is configured to generate two interfaces. An AXILite slave interface is used to load filter coefficients and send commands to the engine to enable the execution of the forward and inverse transform

and it uses the GP AXI 32-bit port. An AXIM interface is used to load and store pixel and transformed data using the hardware implemented memcpy function through the ACP port. The ACP port was selected instead of the HP AXI ports because it offers data coherence with the CPU cache that is needed in the fusion algorithm that shares data between the CPU and the accelerator. An alternative to the custom DMA engine will be to use an available DMA IP core such as the VDMA that also can handle 2D pixel data. This option was not used due to the additional complexities of dealing with code that is hidden from the user. The Elongate flow needs IP that has source code available in VHDL or Verilog that can be used as input into the flow shown in Fig. 1. Fig. 7 shows a section of the code corresponding to the forward wavelet transform filters synthesized by the Vivado HLS tools. Similar code computes the inverse transform filters. Notice that only the filters are computed in hardware and the rest of the wavelet transform is done in software by the CPUs. The filter operations performed by the hardware are shown as equations in (1) and (2):

$$lpAcc[n] = \sum_{k=1}^{12} coeff\_register\_lp[k] input\_b[n-k] \quad (1)$$

$$hpAcc[n] = \sum_{k=1}^{12} coeff\_register\_hp[k] input\_a[n-k] \quad (2)$$

```
   // read data
1  memcpy(buff_in,(float *)(memory+in_offset),(outwidth*2+12)*sizeof(float));

   wav_engine_master_label0:for (int i=0;i<(outwidth+6);i++)
   {

4          input_a=(data_t)buff_in[i*2];
5          input_b=(data_t)buff_in[i*2+1];

           hpMult=coeff_register_hp[0]*shift_register[0];
           lpMult=coeff_register_lp[0]*shift_register[0];
           hpAcc=hpMult;
           lpAcc=lpMult;

10         wav_engine_master_label1:for (int j=1;j<11;j++)
           {
                  lpMult=coeff_register_lp[j]*shift_register[j];
                  hpMult=coeff_register_hp[j]*shift_register[j];
                  hpAcc+=hpMult;
                  lpAcc+=lpMult;
                 shift_register[j-1]=shift_register[j+1];
           }
           lpMult=coeff_register_lp[11]*shift_register[11];
           hpMult=coeff_register_hp[11]*shift_register[11];
           hpAcc+=hpMult;
           lpAcc+=lpMult;
22         shift_register[10]=input_a;
23         shift_register[11]=input_b;
24         if (i>5)
           {
                   buff_out[i*2-12]=(float)hpAcc;
                   buff_out[i*2+1-12]=(float)lpAcc;
           }
   }
   // write data
30 memcpy((float *)(memory+out_offset),buff_out,(outwidth*2)*sizeof(float));
```

Fig. 7 Sample code for FPGA synthesis

The memcpy's shown in lines 1 and 30 move data between the external DDR memories and internal Block-RAMs and the *for* loop in line 9 creates the filters with the help of an internal shift register. The final *if* in line 24 makes sure that only the correct outputs are written to the output buffers. New input data is loaded into the

shift registers in lines 22 and 23 and extracted from the input buffers in lines 4 and 5. Additional pragmas (not shown in Fig. 7) are used to ensure that the tool adds the require AXI interfaces and pipeline registers to obtain an initialization interval of one clock cycle so a new input enters the pipeline in each clock cycle. Notice that the Vivado HLS tool in version 2015.4 does not pipeline the memcpy and it needs to complete before the loop processing can start. It is important to note that all the logic required to implement these functions is created on the PL side by Vivado HLS. Control variables not shown in this sample code activate one of three possible modes that correspond to 1) filter coefficient loading, 2) forward transforms and 3) inverse transform. With this setup, we wrote a kernel level Linux driver to allocate memory that can be accessed by the accelerator with physical addresses and by the processor with virtual addresses. The driver uses the standard "memcpy" function, implemented in this case in software at the user level, for data transfer. For this to work, it is necessary to obtain the physical addresses at which the memory is created by the "kmalloc" calls in the kernel driver, and then use the memory-map calls "mmap" to obtain remapped virtual addresses in user space that can be used by standard "memcpy". Additionally, the Linux driver implements the "ioctl" function, which can be used to control how the data movements take place. In our case, we used this to create different read and write offsets to the kernel allocated memory. The input and output buffers have a size of 4096 32-bit, divided into two areas of 2048 32-bit, which is suitable for an image width up to 2048 pixels. The two buffer areas allow the ARM to prepare the next block of pixels while the hardware engine is processing the second buffer. The double buffering mechanism is illustrated in Fig. 8.

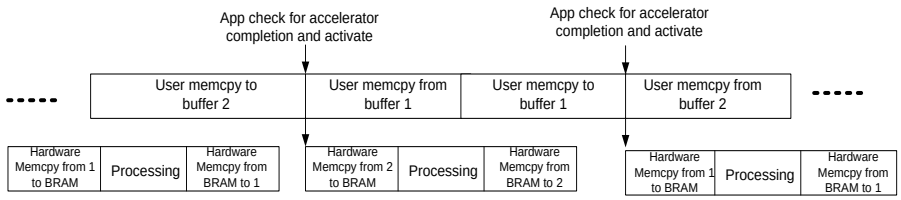

Fig. 8. Double buffering implementation

The input into the flow of Fig. 1 is a C++ description of the forward and inverse DT-CWT including memcpy that are initially compiled with Vivado HLS tools to obtain a netlist in either VHDL or Verilog format. This netlist is then synthesized by the Synplify synthesis tool to obtain a new VHDL netlist based on the implementation primitives available in the target technology. These initial synthesis steps are required to obtain the netlist that will be processed by Elongate. The need for this initial pre-processing is because the Elongate transformation does not take place at source level directly. The reason is that slight changes in the source can have a large effect on timing and also because it is possible to annotate the critical paths found after static timing analysis with the physical flip-flops in the netlist. The timing information is critical to allow Elongate to replace the end-point flip-flops in the critical paths with new soft-macro flip-flops that incorporate the in-situ detection logic. Each primitive flip-flop component in the technology library has a corresponding soft-macro flip-flop stored in the Elongate component library with identical functionality. Part of the user constraints input to Elongate indicate the level of coverage requested for the critical paths in the design. The coverage must be sufficient so that the critical paths of the final design have as endpoints the newly inserted soft macro flip-flops. To determine how many end-points should be protected with in-situ detectors an experiment has been conducted in which they are inserted in blocks of 50. The results are shown in Fig. 9 in which all detectors correspond to logic and not memory. After the insertion the static timing analysis tool is re-run to verify that the critical paths correspond paths with in-situ detectors as end points. The static timing analysis indicates that the runs with 100 and 150 detectors verify this constraint while this is not the case in the run with 50 detectors. The addition of the detectors results in a largely unaffected critical path while complexity increases by around 5% taken into account the control logic that monitors tinhe state of these detectors. Table I shows the resource usage of this hardware wavelet engine after the insertion of 100 timing detectors that is selected as the first point that meet the end-point constraints and has less overhead.

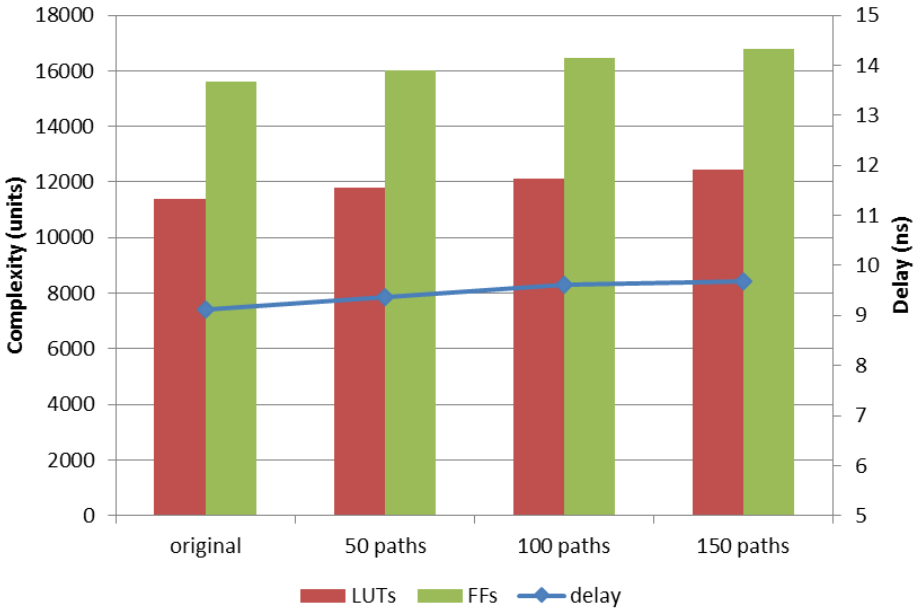

Fig. 9. Logic in-situ detection analysis

TABLE I. IMPLEMENTATION COMPLEXITY OF WAVELET ENGINE

Wavelet Engine	**Implementation Complexity Part: *xc7z020clg484-1***		
	Unitization	***Available***	***Percentage***
Registers	16463	106400	15%
LUTs	12110	53200	22%
DSPs	118	220	53%
BRAMs	4	140	3%

The system correct functionality is verified fusing infrared and natural data obtained from a security application. Sample frames are shown in Fig. 10 that shows two input frames on the left and the resulting frame on the right. The output pixels are then compared with an expected output data stream and no errors are found.

Fig. 10 Demonstration of the fusion system

5.2 Microblaze application

The second user case consists of a 4 Microblaze processors that are implemented in the configurable logic of the Zynq device. Fig. 11 shows the user IP block for the Microblaze system where all the processors use the same configuration consisting of a 5-stage pipeline, no cache, hardware support for 32-bit multiplier, divider, shifter and comparator and extended floating-point support. Table II reports the logic utilization of this system. The processors access instructions and data from BlockRAM memories and the critical paths go from the processor logic to the BlockRAM as verified in the timing analysis reports. Notice that in other configurations that include cache memories inside the Microblaze this will change and paths could extend to these internal memories. The Microblaze processors are made available by the vendor as encrypted netlists and this will limit the applicability of the Elongate flow that needs access to unencrypted netlists to be able to modify them and insert the detectors. Because in the considered configuration the critical paths exit the processor logic and arrive to BRAM memories the flow can still be used if the BRAM paths can be adequately protected with the modified detector shown in Fig. 2.b. The new flow for BRAMs uses two clocks with different clock phases but the same frequency as described in Section 3. The detectors are inserted protecting the data and address lines to the BlockRAMs and communicate with the EFF (embedded flip-flop) control visible in Fig. 6. Data and binaries are copied to the BRAM by the ARM host processor through the host AXI directly writing the BlockRAMs. Once data and binary copying completes the host processor activates the Microblaze by removing the reset signal. The Microblaze processors proceed to execute the code stored in the BlockRAMs and issue an interrupt to the host once the processing completes. A UART is also available so the processors can report on execution progress. Each processor can execute a different binary or the same binary with different data sets.

In this system the amount of memory implemented per Microblaze is 32 KB which means that only 15 address lines (out of a maximum number of 32) need to be protected. The full 32 data lines are protected so that each Microblaze protects a total of 47 paths corresponding to 32 data lines and 15 address lines. This results in a total of 188 (47x4) paths protected and an additional complexity of 14% compared with the original design. Notice that in this case study we are protecting almost double the number of paths and also the complexity of the memory sensors is higher than the logic sensors because three flip-flops are added corresponding to SFF, MFF and synchronizer flip-flop plus the combinatorial logic while in the logic sensor MFF replaces one flip-flop already present in the original design. It is possible to reduce this overhead if only the slowest Microblaze is protected instead of all of them to approximately 4%. This however will not be as reliable as protecting all processors since if the processors run different applications or even the same application with different data inputs there is the possibility that the critical paths are not exercised in the same way.

TABLE II. IMPLEMENTATION COMPLEXITY OF MULTICORE MB

MB system	**Implementation Complexity Part: *xc7z020clg484-1***		
	Unitization	***Available***	***Percentage***
Registers	8624	106400	8%
LUTs	11868	53200	22%
DSPs	25	220	11%
BRAMs	32	140	22%

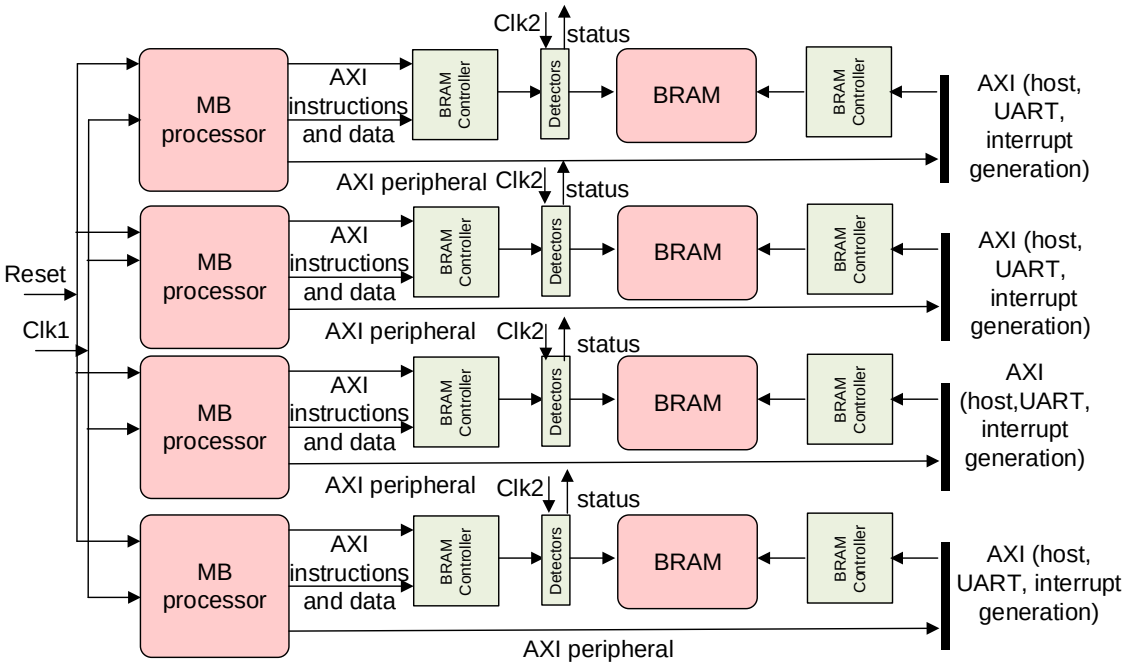

Fig. 11. Microblaze multi-core hardware

6 Power and Performance Analysis

This section compares the power consumption and performance of the two case studies implemented in the FPGA after applying voltage and frequency scaling. Fig. 12 shows the maximum frequency that can be maintained at each voltage which is equivalent to the point in which the in-situ detectors are activated. This represents an optimal pair of voltage/frequency. It can be noted that both hardware blocks exhibit a very similar profile. The static timing analysis obtained from the vendor tools at 1 volt for both cores reports a worst case data path delay of 8.8 ns for the Microblaze system and 9.4 ns for the wavelet system which are relatively close and this could explain why the voltage/frequency lines are close to each other in Fig. 12. In any case this relation is totally dependent on the timing characteristics of the cores and could be very different for other cores. In both cases we can observe that although the timing analysis reports a performance level close to 100 MHz the real system achieves close to 170 MHz at 1 volt as seen in Fig.12.

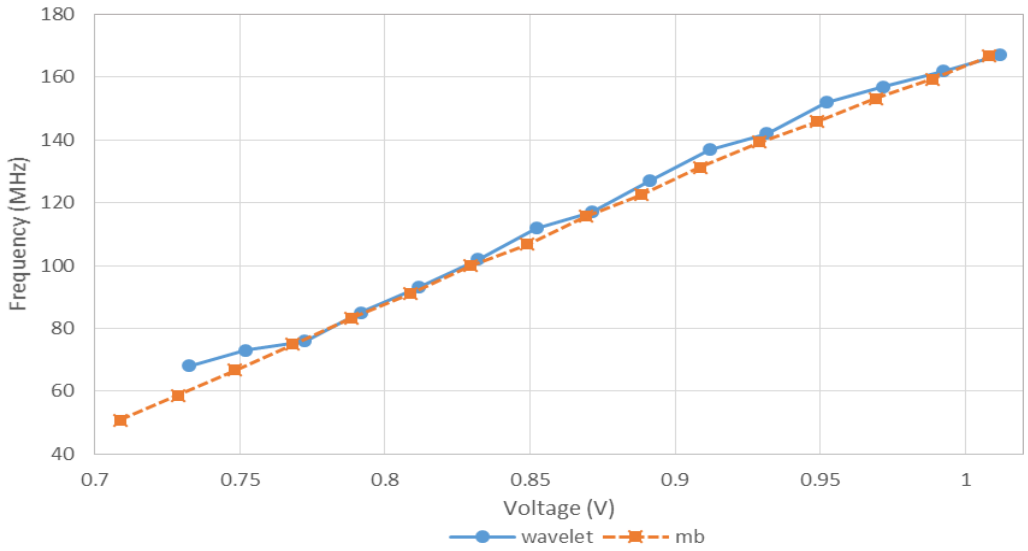

Fig. 12. Frequency and voltage analysis for both systems

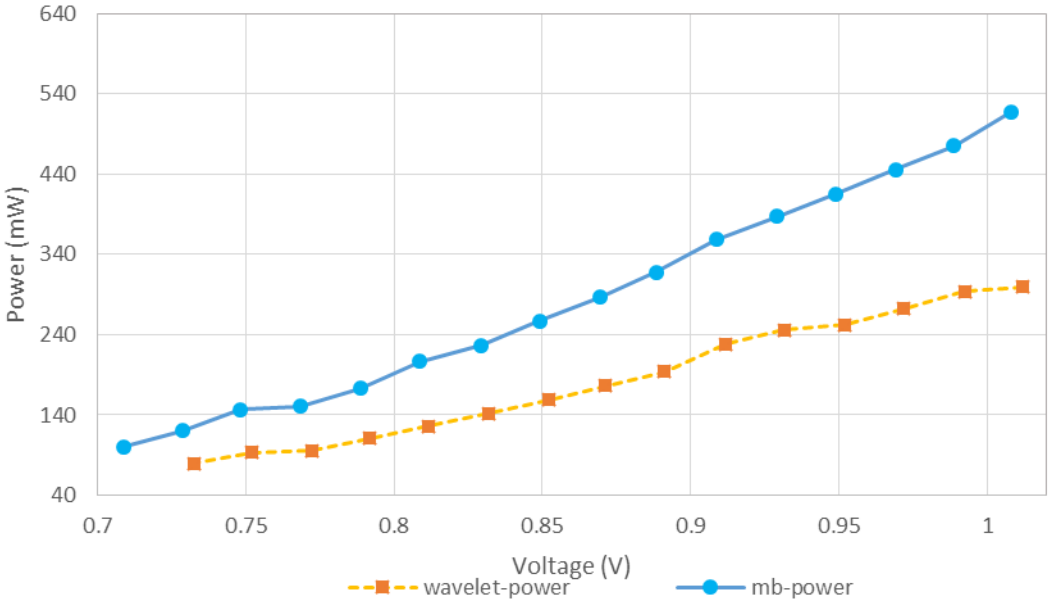

Fig. 13. Voltage and power analysis for both systems

Fig.13 shows the power analysis at each voltage/frequency point. It is clear that power in the Microblaze system is considerable higher than in the wavelet engine. Both systems use a similar number of LUTs although the wavelet system uses a significant larger number of registers to implement a deep pipeline. It is possible that this deep pipeline results in a more power efficient design since fewer glitches are allowed to propagate through the combinatorial logic. All the power measurements correspond to the Vccint rail that supplies power to the FPGA fabric and directly affects the speed of propagation of the signals in the fabric but not to Vccaux that powers the transistors that form the BlockRAM memory arrays. In this research Vccaux is not modified since the detectors cannot protect these transistors. The amount of power due to the Vccaux rail remains constant. In both cases the

figures confirm that operating at lower points of voltage and frequency result in significant power reductions in the Vccint rail.

7 Energy Analysis

Reducing the operating voltage/frequency point increases the execution time of the algorithm and since energy is the product of power and time the overall energy effect could be detrimental. To understand the energy effects the cores have been deployed in two different operational scenarios.

7.1 Wavelet energy in fusion system

The wavelet represents 70% of total computation time and it is important to note that while the accelerator is running, the software needs to prepare the next group of pixels and this task is done by the ARM that must move data to an area that can be accessed by the hardware engine. Once the data movement is completed the thread running on the ARM processors activates the accelerator and starts preparing the next group of pixels using the double buffering mechanism described in Fig. 8. The investigation shows that the accelerator finishes processing the supplied data before the processor can prepare the next group of pixels. This shows that the data movement done by the processor is the performance limiting factor and not the hardware computation. This suggests that if we run the hardware slower we can maintain the same level of performance but we can save power and energy by decreasing the frequency and voltage levels. Fig. 14 shows how reducing the FPGA voltage from nominal 1 volt to a minimum of 0.72 volts reduces the energy requirements. The amount of time required to complete the DT-CWT transforms for each of the voltage points increases but because this time remains below the time required by the processor to prepare the next group of pixels there is no negative effect in performance. The lowest point of 0.72 volts that corresponds to a frequency of 68 Mhz (as seen in Fig. 12) is sufficient to support the operation within the time constraints imposed by the processor. The energy requirements at this point are 75% lower than at maximum frequency.

Fig. 15 compares executing the wavelet transforms and the rest of the fusion algorithm using only the processor with hardware acceleration of the transforms using the FPGA device. Once the accelerators start working the processor power (PS) reduces since it does not need to perform the wavelets but the FPGA fabric power (PL) increases resulting in an overall increase of the power consumption as seen in Fig. 15 compared with the software only solution. However processing time reduces by a factor of 3 and this means that there is a similar reduction effect on energy. Fig. 15 shows that overall energy is reduced from 810 mJ to 258 mJ which is approximately 70% with the combination of hardware acceleration and voltage scaling. In this system the wavelet hardware does not use interrupts to the ARM processor and it receives commands from the processor using an AXILite interface while the pixel data is moved using AXI DMA accesses. The ARM processor reads the core status using registers mapped in the AXILite interface to check for accelerator completion once it has finished preparing the pixels for the next frame. If the register indicates that the wavelet engine has completed its task while the processor was preparing the next set of pixels, the processor knows that the core is running too fast and it can then slow down the accelerator and reduce the voltage for the next iteration. This process repeats until the completion of the ARM tasks and the wavelet hardware take place approximately at the same time.

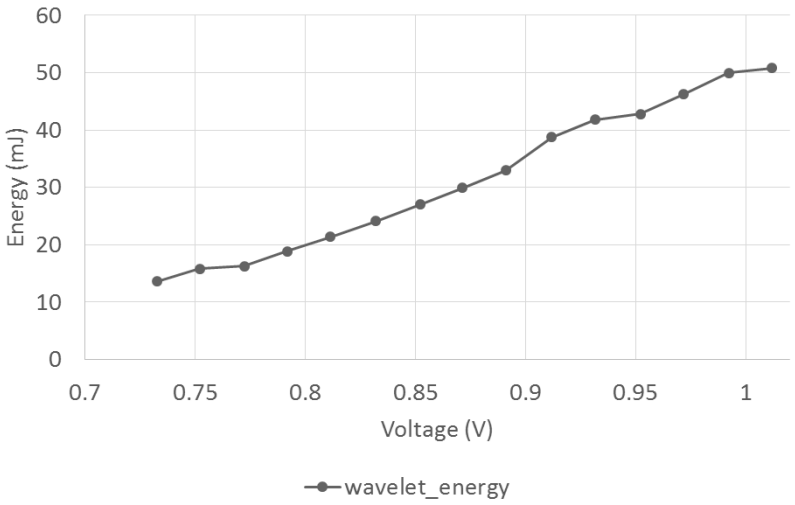

Fig. 14. Voltage and Energy analysis in the wavelet hardware

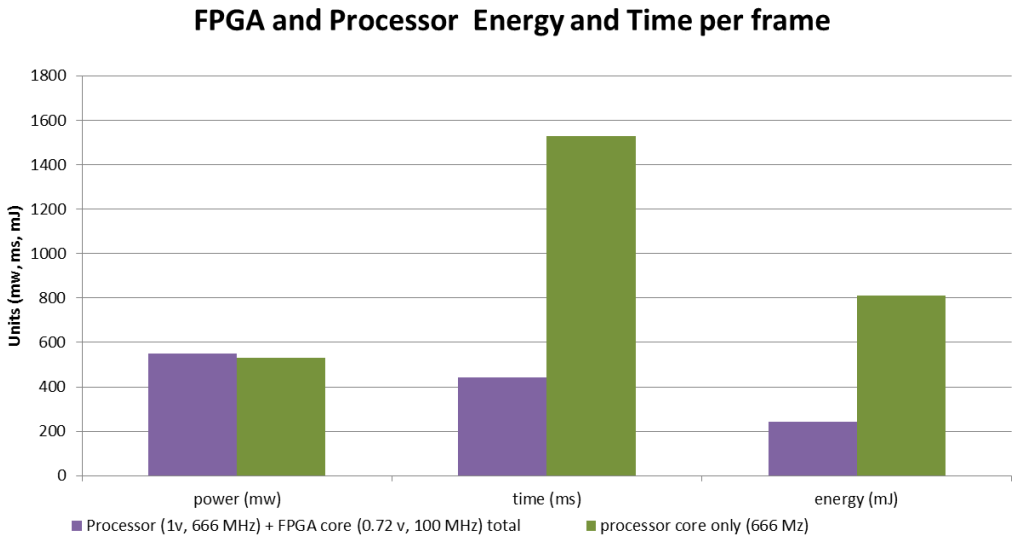

Fig. 15. Total power and energy for the wavelet system

7.2 Microblaze energy

The second case study focuses on a cluster of Microblaze processors that behave as an overlay architecture for the FPGA device and that can be programmed using standard C/C++ code instead of custom logic as done with the wavelet hardware. In this case the ARM processor moves data and binaries to the local BlockRAM memories and activates the Microblaze processors. The ARM processors enter into the sleep mode and wait to receive the wake-up interrupt from the Microblaze cores. Since they do not actively participate in the benchmark computation the power and energy requirements of the ARM processor during sleep are ignored. The Microblaze processors issue interrupts to the ARM processor to wake it up from sleep and to indicate completion. The ARM processor can use this information together with the available time budgets to proceed to increase or decrease the performance of the Microblaze cores. To evaluate the energy effects in this situation we have selected *linpack* as the benchmark that the Microblaze processors execute. We have also measured the static energy costs of the FPGA assuming that the ARM processor can clock gate the Microblaze's once they complete their task until the

next task is required. This experiments measure if it is better to run fast and clock gate compared with run slow at a lower voltage and frequency point. The experiments assign a time budget defined by the slowest execution and then consider if running faster is beneficial from an energy point of view. Configurations that execute faster than the time budget clock gate the hardware so that only leakage energy remains until the time budget is exhausted. Fig. 16 shows the total power and the static power with gated clocks while Fig. 17 shows the resulting total, static and computation energy needed to complete the *linpack* benchmark. It also shows the amount of time in which the processors are clock gated because they have finished the computation. The static energy corresponds to the energy used during the clock gated time, computation energy corresponds to the energy needed to run the benchmark and contains both dynamic and static components. Total energy is the summation of computation and static energy. Each processor executes its own copy of the benchmark and each voltage point in the *x* axis corresponds to a different frequency point as shown in Fig. 12. The fastest execution corresponds to the 1 V point in which the Microblazes run at 166 Mhz and remain during the longest period in the clock gated state. The point at 0.7 volts is the slowest execution in which the Microblazes run at 50 MHz and did not use the clock gated state. The energy usage at 1 V is 112 mJ while at 0.70 V is 54 mJ which corresponds to a 60% energy reduction. This experiment confirms that despite the reduced static energy thanks to clock gating the slow execution at minimum voltage is still significantly more energy efficient than the fast execution followed by the clock gated state.

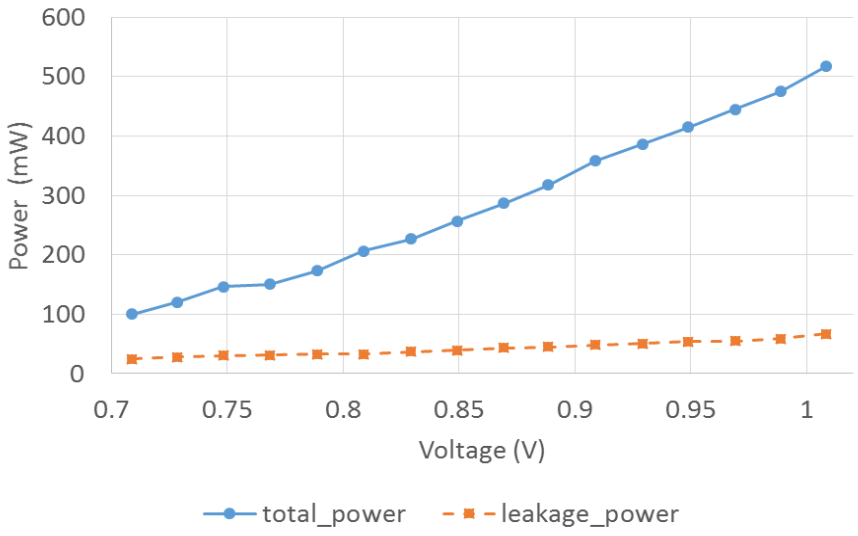

Fig. 16. Voltage and power analysis in the Microblaze system

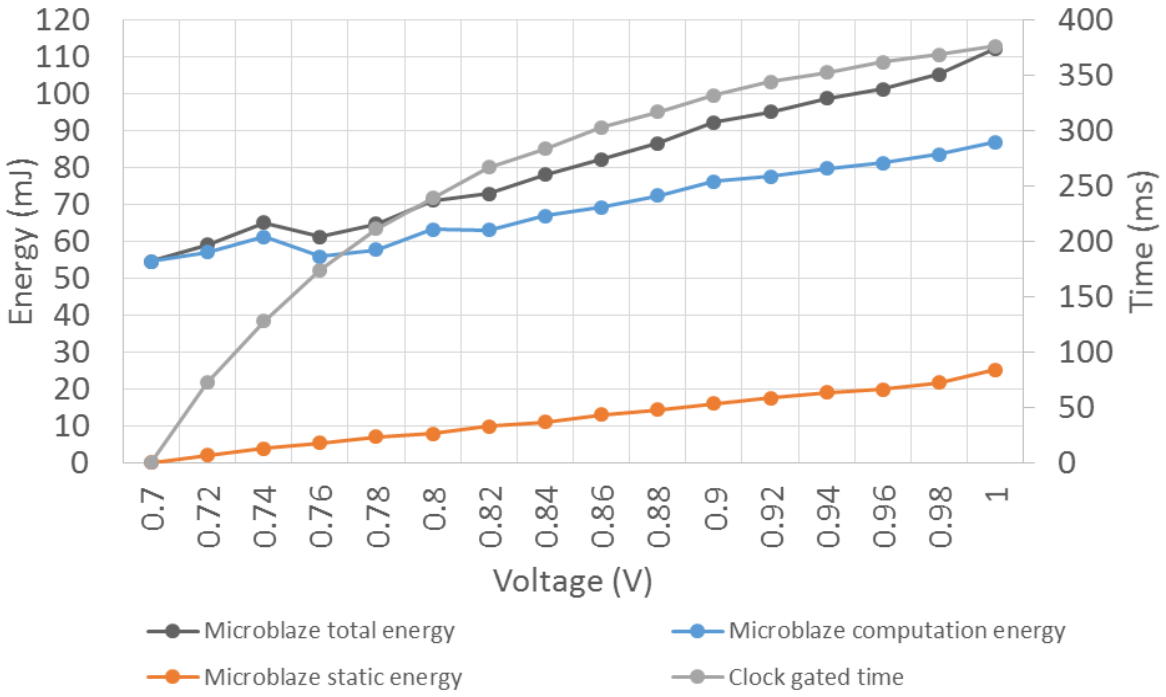

Fig. 17. Voltage and energy analysis in the Microblaze system

8 Conclusions and Future Work

This paper has presented an energy proportional system based on memory/logic in-situ detectors and voltage/frequency scaling. The system has been applied to two case studies based on custom logic generated using a high level synthesis tool and an overlay architecture formed by Microblaze soft processors. The first case study shows the acceleration of a video fusion application based on wavelet hardware while the second scenario uses standard soft Microblaze processors that are programmed using C code. The performance, power and energy consumption of each of these systems has been measured at different points of voltage and frequency. Overall we observe that the technology allows stretching the silicon to deliver energy reductions between 60 to 70% at minimum voltage depending on the implementation approach. Alternatively the silicon can be stretched for performance in which case approximately 70% better performance can be obtained at nominal voltage compared with the frequency value obtained from the tools. The extension of the original flip-flop end-points to the BlockRAM end-points is done with the addition of phased clocks and a redesign of the in-situ detectors. As future work we would like to further automate the insertion of memory and logic sensors in the hardware so that the technology can be used transparently and without detail knowledge of the FPGA architecture. We would like also to develop a scheduling algorithm that will enable the ARM processor to automatically assign points of voltage and frequency to the FPGA fabric according to performance demand requirements. The interested reader can check http://www.bris.ac.uk/engineering/ research/microelectronics/enpower/ for demonstration videos.

Acknowledgements: We would like to thank the support received from EPSRC for this work which is part of the ENPOWER (EP/L00321X/1) and the ENEAC (EP/N002539/1) projects.

References

1. C. Zhang, R. Chen and V. K. Prasanna, High Throughput Large Scale Sorting on a CPU-FPGA Heterogeneous Platform, 23nd Reconfigurable Architectures Workshop (RAW), 30th IPDPS, Feb 2016
2. A. Putnam et al., "A reconfigurable fabric for accelerating large-scale datacenter services," 2014 ACM/IEEE 41st ISCA, Minneapolis, MN, 2014, pp. 13-24.
3. Enabling Coherent FPGA Acceleration, http://openpowerfoundation.org/blogs/enabling-coherent-fpga-acceleration/
4. J. Luis Nunez-Yanez, M. Hosseinabady and A. Beldachi, "Energy Optimization in Commercial FPGAs with Voltage, Frequency and Logic Scaling," in IEEE Transactions on Computers, vol. 65, no. 5, pp. 1484-1493, May 1 2016
5. A. Rahman, S. Das, T. Tuan, and A. Rahut, "Heterogeneous routing architecture for low-power FPGA fabric," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 183–186, 2005.
6. J. Ryan, and B. Calhoun, "v," *IEEE Custom Integrated Circuits Conference* (CICC), pp.1–4, 2010.
7. F. Li, Y. Lin, and L. He, "Vdd programmability to reduce FPGA interconnect power," in *IEEE/ACM International Conference on Computer Aided Design* (ICCAD'04), pp.760-765, 2004.
8. F. Li, Y. Lin, L. He, and J. Cong, "Low-power FPGA using pre-defined dual-Vdd/dual-Vt fabrics," in *Proceedings of 12th International Symposium on Field Programmable Gate Arrays*, (FPGA'04), pp.42-50, 2004.
9. A. Raham and V. Polavarapuv, "Evaluation of low-leakage design techniques for field programmable gate arrays," in *Proceedings of 12th International Symposium on Field Programmable Gate Arrays* (FPGA'04), pp.23–30, 2004.
10. J. Lamoureux, and S. Wilton, "On the interaction between power-aware FPGA CAD algorithms," in *International Conference on Computer Aided Design (ICCAD'03)* pp.701-708, 2003.
11. J. Lamoureux, and S. Wilton, "Clock-aware placement for FPGAs," in *Field Programmable Logic and Applications,* (FPL'07), pp.124 –131, 2007.
12. A. Gayasen, et al. "Reducing leakage energy in FPGAs using region constrained placement," in *Proceedings of the 12th International Symposium on Field Program*
13. Information available at http://www.xilinx.com/support/documentation/application_notes/xapp555-Lowering-Power-Using-VID-Bit.pdf
14. Information available at http://www.altera.com/literature/wp/wp-01200-power-performance-zettabyte-generation-10.pdf
15. C. Chow, L. Tsui, P. Leong, W. Luk, and S. Wilton, "Dynamic voltage scaling for commercial FPGAs," in *Proceedings of IEEE International Conference on Field-Programmable Technology*, pp.173-180, 2005.
16. N. Atukem, J. Nunez-Yanez, "Adaptive voltage scaling in a dynamically reconfigurable FPGA-based platform," *ACM Trans. Reconfigurable Technol. Sys*t. Vol.5, no. 4, 2012.
17. S. Das, et al., "Razor II," *IEEE J. Solid-State Circuits*, pp. 32-48, Jan. 2009.
18. J. M. Levine, E. Stott, and P. Y. K. Cheung, "Dynamic voltage & frequency scaling with online slack measurement," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (FPGA'14), pp. 65-74, 2014.
19. Nunez-Yanez, J.L., "Adaptive Voltage Scaling with In-Situ Detectors in Commercial FPGAs," Computers, IEEE Transactions on , vol.64, no.1, pp.45,53, Jan. 1 2015
20. Nunez-Yanez, JL, 2016, 'Computing to the Limit with Heterogeneous CPU-FPGA Devices in a Video Fusion Application'. ARC, 2016 Brazil, March 22–24, Proceedings. Springer Verlag, pp. 41-53
21. M. Wirnshofer, et. Al , "Adaptive voltage scaling by in-situ delay monitoring for an image processing circuit," 2012 IEEE 15th DDECS, Tallinn, 2012, pp. 205-208.
22. Hill, P, Bull, D & Canagarajah, C 2005, 'Image fusion using a new framework for complex wavelet transforms'. IEEE ICIP 2005, Genova, Italy. IEEE, pp. II-1338 - II-1341